ALGORITHMIC DIFFERENTIATION APPLIED TO THE OPTIMAL CALIBRATION OF A SHALLOW WATER MODEL

Félix DEMANGEON[(1)], Cédric GOEURY [(2)]
Fabrice ZAOUI[(2)], Nicole GOUTAL[(1) (2)]
Valérie PASCUAL[(3)], Laurent HASCOËT[(3)]

[(1)]Laboratoire d'Hydraulique Saint-Venant (entité commune à EDF R&D, CEREMA et l'Ecole des Ponts), Chatou, France

[(2)]EDF R&D, Laboratoire National d'Hydraulique et Environnement (LNHE), Chatou, France - e-mail: fabrice.zaoui@edf.fr

[(3)]INRIA, (ECUADOR team), Sophia-Antipolis, France

Les informations de sensibilité fournies par les dérivées sont indispensables en science dans de nombreux domaines. En analyse numérique, calculer très précisément la valeur des dérivées d'une fonction d'un simulateur physique peut relever du défi. La méthode classique des Différences Finies (DF) est une solution simple à mettre en œuvre pour estimer la valeur d'une dérivée. Cependant, elle reste très sensible numériquement et coûteuse en temps de calcul. A contrario la méthode de la Différentiation Algorithmique (DA) est une aide puissante pour le calcul des dérivées d'une fonction décrite au moyen d'un programme informatique. Quelle que soit la complexité des algorithmes mis en œuvre dans l'expression d'une fonction, elle calcule précisément sa dérivée en minimisant les efforts de développement.
Cet article montre l'apport de la DA en comparaison des DF sur le problème du calage d'un modèle hydraulique à surface libre 1D de classe industrielle. Le calage du modèle est réalisé par un optimiseur mathématique déterministe nécessitant le calcul précis de la sensibilité de la cote d'eau par rapport au frottement sur le fond de la rivière. Deux cas tests réels de comparaison sont présentés. Ils permettent de valider la supériorité attendue de la DA comme outil d'aide à l'obtention d'un calage optimal.

KEY WORDS: Modèle Hydraulique à surface libre, Calage, Différentiation Algorithmique

Algorithmic Differentiation for the optimal calibration of a shallow water model

The information on sensitivity provided by derivatives is indispensable in many fields of science. In numerical analysis, computing the accurate value of the derivatives of a function can be a challenge. The classical Finite Differences (FD) method is a simple solution to implement when estimating the value of a derivative. However, it remains highly sensitive numerically and costly in calculation time. Conversely, the Algorithmic Differentiation Method (AD) is a powerful tool for calculating the derivatives of a function described by a computer program. Whatever the complexity of the algorithms implemented in the expression of a function, AD calculates its derivative accurately and reduces development efforts.
This article presents the contribution of AD in comparison to FD in the problem of calibrating an industrial class 1D shallow water model. Model calibration is performed by a deterministic mathematical optimiser requiring accurate calculation of the sensitivity of the water surface profile in relation to the friction on a river bed. Two comparative real test cases are presented. They permit validating the better performance expected from AD as a tool used to obtain optimal calibration.

KEY WORDS: Shallow Water Model, Calibration, Algorithmic Differentiation

I INTRODUCTION

Computing sensitivities of a numerical model is a key ingredient in Scientific Computing. The derivatives that express these sensitivities must be computed with the best possible accuracy for applications such as inverse problems, data assimilation, optimisation, or Uncertainty Quantification. When the model is given as a computer program, several options exist to obtain its derivatives. Finite Differences (FD) is easily implemented but returns approximate derivatives whose poor accuracy will degrade performance of the complete application. A much better option is to create a new program that computes the exact, analytical derivatives of the model. This new program can be written "by hand", but this is a long and error-prone

process. Keeping the derivative model up to date with modifications of the original model is cumbersome. Alternatively, Algorithmic Differentiation (AD) [Griewank and Walther, 2008] is a way to automate the creation of the derivative program, thus providing accurate derivatives for a minimal development effort. This article presents the application of the TAPENADE AD tool [Hascoët and Pascual, 2013] developed by INRIA, on the 1D shallow water model MASCARET [Goutal et al., 2012]. It validates the use of the differentiated code for an inverse problem of parameter estimation.

Regarding shallow water programs like MASCARET, the type of river bed is modelled by a friction coefficient. This coefficient takes into account the friction of the fluid on the bottom as well as other phenomena not modelled elsewhere such as turbulence and channel bends. This article presents the application of AD to an inverse problem, namely the automatic calibration of the friction coefficient based on a water surface profile measured during flooding at steady discharge. This permits testing both the AD TAPENADE software with an industrial calculation code and improving the current method used to calibrate the friction coefficient in MASCARET.

Initially, after having introduced the MASCARET 1D water flow calculation, the problem of calibrating the friction coefficient is presented (cf. Section II). Section III focuses on the presentation of the optimisation algorithm chosen to carry out this task. Then, Section IV describes the principle and use of AD. In Section V, the results obtained from the different test cases are presented and discussed. Finally, the conclusions and outlook are presented in Section VI.

II THE WATER MODELLING SYSTEM

The calculation code on which the works presented here is based is the MASCARET 1D free surface modelling software. This software is part of the open-source hydro-computing system TELEMAC-MASCARET (www.openmascaret.org). Applications of this system are many, from flood modelling and flood plain modelling to the calculation of flood waves resulting from dam breaches, sediment transport and water quality. It is composed of three main kernels: steady subcritical flow, unsteady subcritical flow and unsteady super-critical flow. The steady subcritical flow kernel is the target of this study.

II.1 Saint-Venant 1D equations

As mentioned above, we focus only on 1D steady flows. The equations governing this type of flow are 1D Barré St-Venant equations given by the following formulations [Chanson, 2004]:

$$\frac{\partial Q}{\partial x} = q_a \tag{1}$$

$$\frac{\partial}{\partial x}\left[\frac{Q^2}{A}\right] + gA\frac{\partial Z}{\partial x} = gA(S_f + S_s) + q_a \tag{2}$$

where A represents the wetted section (m^2), Q the flow rate (m^3/s), q_a the inflow, Z the free water surface (m), S_f the head losses resulting from the friction of the fluid on the walls, S_s the singular head losses (narrowing, etc.).

Term S_f of the momentum equation (2) is calculated using empirical friction laws such as that of Strickler (19^{th} century) whose relation is given below:

$$S_f = \frac{Q^2}{K^2 A^2 R^{4/3}} \tag{3}$$

where R denotes the hydraulic radius (wetted surface divided by the wetted perimeter in m) and K, the Strickler coefficient modelling the nature of the channel bed ($m^{1/3}/s$).

In the MASCARET hydraulic calculation, the river bed is divided into two zones, the main channel and the flood plain. The main channel is the main flow zone. The flood plain is the secondary zone: this zone participates in the flood flow, though it has a specific friction coefficient due to the different natures of the soils. This coefficient is defined by the cross-section and by the bed, and is constant per friction zone, whose sizes and numbers are determined by the study data. The friction coefficient takes into account the friction of the walls on the fluid and the dissipation phenomena not modelled elsewhere (turbulence, etc.). Thus it cannot be determined directly by the study data and must be adjusted using the water surface profiles

measured for a given flow rate. Done manually, this model calibration step is time consuming, but it is indispensable to ensure the quality of the study. That is why an automatic calibration method using measured elevation data has already been integrated in MASCARET.

This automatic calibration method is based on a first order unconstrained optimisation method known as "gradient descent optimisation", with a gradient approximated by finite difference. The disadvantages of this approach are:

- The velocity of convergence to the minimum sought can prove slow.
- In certain cases, its accuracy can be poor.
- The values of the friction coefficients obtained can exceed the values prescribed.

Therefore the objective of this work is to propose a more efficient automatic calibration algorithm capable of eliminating the limitations mentioned, using Algorithmic Differentiation methods.

II.2 Automatic calibration

Automatic calibration is an inverse method used to find an "admissible" constant friction coefficient K per zone, resulting in the calculation of a water surface profile close to the water surface profile measured for a steady flow. The optimal search for this coefficient is done by minimising a cost function calculating the difference between the level computed by the numerical model and the measured level:

$$J(K) = \sum_{c=1}^{N_{flood}} \left[\sum_{j=1}^{N_{meas}^{c}} \left(\alpha_j^c \left(Z_j^{meas} - Z_j^{calc}(K) \right)^2 \right) \right] \tag{4}$$

where N_{flood} represents the number of floods, N_{meas}^{c} the number of measures linked to these floods, α_j^c a weighting coefficient that can be set to less than 1 when a measure is deemed uncertain, Z_j^{meas} the water level measured at point j and Z_j^{calc} the height calculated by the model.

Generally, optimisation methods are used to solve minimisation problems. The former can be very different according to the form of the function to be minimised (convex, quadratic, nonlinear, etc.), its regularity and the dimension of the space studied [Nocedal and Wright, 2006]. Many deterministic optimisation methods are known as gradient descent methods, among which the best known is the Newton method, which is the approach used in this work [Gilbert and Lemaréchal, 1989]. This minimisation process will quickly find, when successful, a better solution in comparison with the value of the initial guess.

III THE BFGS QUASI-NEWTON METHOD

As mentioned above, the optimisation method chosen to minimise the cost function (4) is based on the application of the Newton method to the gradient of the functional J, to find the zeros and then the extremes, thus the minimum. This method involves the calculation of the first and second derivatives of the cost function. The main disadvantage of this type of approach is the use of the second derivative (Hessian) of the cost function $\nabla^2 J(K)$. Indeed, at each iteration of the Newton algorithm, it is necessary to calculate the Hessian and solve a linear system of the matrix $\nabla^2 J(K)$. For large problems, the resolution of the linear system is out of reach. An alternative is to use algorithms such as the Quasi-Newton algorithm which provides Hessian approximations that improve as the iterations progress, for a reasonable cost. Therefore the method chosen to perform this work is the constrained Broyden Fletcher Goldfarb Shanno Quasi-Newton Method (BFGS).

Using a constrained optimisation method makes it possible to impose boundaries when seeking the parameter to be calibrated. Although the friction coefficient takes into account dissipation phenomena that cannot be represented in the numerical model, it is directly dependent on the type of surface composing the river bed. Thus an interval of acceptable "physical" values exists for searching the friction coefficient as a function of the type of soil.

The optimisation method used involves calculating the gradient of the cost function. One could compute an accurate gradient by manually differentiating the calculation code. However this would be time-consuming and error-prone, as this implies writing a code of a size similar to the original code (more than 10,000 lines

of FORTRAN for MASCARET steady flow kernel). Nonetheless, AD software tools can alleviate this cost [Griewank and Walther, 2008].

IV ALGORITHMIC DIFFERENTIATION

Algorithmic Differentiation of programs is a powerful technique for evaluating the derivatives of functions described by computer programs. Contrary to traditional approaches, such as derivation by finite differences, AD provides accurate derivatives at a relatively cheap cost, for a simple mathematical formula as well as for a program with more than 100000 lines of code. By calculating an exact derivative, AD thus plays a key role in developing a new optimisation method for the automatic calibration of the friction coefficient. This section describes the principle of AD. Then, after having presented the TAPENADE software [Hascoët and Pascual, 2013], used to differentiate the kernel of the steady calculation of MASCARET, its application is described in the framework of implementing the optimal calibration of a shallow water model [Cunge *et al.*, 1980; Fread and Smith, 1978].

IV.1 Principle of Algorithmic Differentiation

Every calculation program, or at least its run-time trace, can be seen as a sequence of assignments involving only unary (trigonometric function, square root, etc.) and binary (additions, multiplications, etc.) operations. Therefore, the program can be seen as a composition of elementary functions, to which one can apply the chain rule of calculus to obtain its derivative with analytic accuracy.

Consider a mathematical function F:

$$\begin{aligned} F : \Re^n &\to \Re^m \\ X &\mapsto Y \end{aligned} \tag{5}$$

Assume F is implemented as a sequence of r computer program instructions, each of them implementing an elementary function. Call f^l the elementary function corresponding to instruction l:

$$Y = F(X) = f^r \circ f^{r-1} \circ ... \circ f^1(X) \tag{6}$$

By applying the chain rule of calculus to equation (6), the Jacobian matrix A of F, which is a $m \times n$ matrix, writes as the product of the derivatives of the f^l.

$$A(X) = \frac{\partial F}{\partial X}(X) = \frac{\partial f^r}{\partial x^{r-1}}(x^{r-1}).\frac{\partial f^{r-1}}{\partial x^{r-2}}(x^{r-2})\cdots\frac{\partial f^2}{\partial x^1}(x^1).\frac{\partial f^1}{\partial X}(X) \tag{7}$$

Explicit evaluation of equation (7) can be extremely costly, as each derivative involved is a matrix roughly of size $q \times q$, where q is the number of elementary variables active at the time of the instruction, i.e. of the order or larger than m and n. Therefore, AD rather focuses on computing two useful projections of A, from which it may even be easier to retrieve the full A if necessary. This results in the two so-called "modes" of AD:

- The tangent mode computes $\delta Y = A.\delta X$ for any arbitrary vector $\delta X \in \Re^n$. In other words, it computes a directional derivative which is the first-order variation of the output for a small variation of the input in direction δX. From equation (7), it is clear that $\delta Y = A.\delta X$ is most efficiently computed from right to left, leading to only matrix-vector products. The total cost of evaluating δY is only a small multiple of evaluating F. If δX is taken as an element of the Cartesian basis, δY is a column of the Jacobian A. The full A can be obtained by repeated calls to the tangent code on the input space's Cartesian basis. Notice that tangent mode implies the same computation order as the original program, and therefore derivative computations can be introduced into the original code by inserting a derivative instruction before each original instruction. Implementation of the tangent mode is therefore straightforward. Table 1 (middle) shows the tangent differentiated code of a simple code shown on the left, which computes the norm of a 3D vector. Derivative variable names are built with an appended "d". One can see the straightforward structure of the tangent code. This tangent code must be run three times to obtain the three components of the gradient.

- The adjoint mode computes $\delta^* X = \delta^* Y.A$ for any arbitrary (transposed) vector $\delta^* Y \in \Re^m$. In other words it computes the gradient with respect to X of a scalar function, which is the weighted sum of all elements of Y with weightings $\delta^* Y$. From equation (7), it is clear that $\delta^* X$ is most efficiently computed from left to right, since this leads only to vector-matrix products. Just like for the tangent mode, the total cost of evaluating $\delta^* X$ is only a small multiple of evaluating F. If $\delta^* Y$ is taken as an element of the Cartesian basis, $\delta^* X$ is a row of the Jacobian A. The full A can be obtained by repeated calls to the adjoint code on the output space's Cartesian basis. Therefore, the adjoint mode tremendously outperforms the tangent mode when $m = 1$, which is the case in most optimization or inverse problems applications. The computation order for the derivatives is reversed from the original program's order, which makes implementation of the adjoint mode a technical challenge (see [Hascoët and Pascual, 2013]). In particular, intermediate values from the original computation must be made available to the derivative computation in reversed order, leading to difficult memory problems and trade-offs [Naumann, 2012]. Table 1 (right) shows the adjoint differentiated code of the same simple code. Derivative variable names are built with an appended "b". One can see the reversed structure of the adjoint code, and the somewhat counter-intuitive shape of the derivative instructions. However, this adjoint code returns the three components of the gradient in only one run.

Table 1: Algorithmic differentiation of a simple code

Original code	Tangent code	Adjoint code
s = x*x	sd = 2.0*x*xd	s = x*x
s = s + y*y	s = x*x	s = s + y*y
s = s + z*z	sd = sd + 2.0*y*yd	s = s + z*z
n = SQRT(s)	y = s + y*y	n = SQRT(s)
	sd = sd + 2.0*z*zd	sb = 0.5*nb/SQRT(s)
	s = s + z*z	nb = 0.0
	nd = 0.5*sd/SQRT(s)	zb = zb + 2.0*z*sb
	n = SQRT(s)	yb = yb + 2.0*y*sb
		xb = xb + 2.0*x*sb
		sb = 0.0

For further details on AD and the associated research, the reader can refer to [Griewank and Walther, 2008], [Naumann, 2012], to the proceedings of the AD conferences, and to the AD community website www.autodiff.org.

IV.2 Algorithmic Differentiation implementation approaches

AD tools rely mostly on two different approaches, Operator Overloading and Source Transformation:

- The Operator Overloading approach is possible only on languages that support overloading, such as C++, ADA and FORTRAN 90. Examples of AD tools of this kind are ADOL-C [Walther and Griewank, 2012], and dco [Lotz et al., 2011]. This AD method is the simplest to implement, since it requires only the definition of a new data-type, and of the overloaded arithmetic operations on this data type. Changes to the original code are minimal. However, efficiency is limited by the run-time overhead of the overloading mechanism, and the reversed nature of the adjoint mode contradicts the natural order of overloading, causing extra run-time and memory overhead;
- The Source Transformation approach is used for instance in the tools TAF [Giering et al., 2005], ADIC [Bischof et al., 1997], ADIFOR [Bischof et al., 1991], OpenAD [Utke et al., 2008], and TAPENADE [Hascoët and Pascual, 2013]. These tools target mainly C and Fortran 90 programs. Source transformation uses concepts from compilation, for example the generation and transformation of abstract syntax trees. Like in a compiler, the source program is parsed, analysed,

and then transformed into a new differentiated source program. Sophisticated compilation techniques allow for optimisations that improve performance of the differentiated code.

IV.3 The TAPENADE AD tool

The AD tool chosen for this work is TAPENADE, developed by the ECUADOR team of INRIA [Hascoët and Pascual, 2013]. Tapenade is based on the source transformation approach. It globally analyses the code to which it is applied and fully rebuilds a differentiated program, adding instructions into the original program. It applies to both FORTRAN 90 and C. TAPENADE provides both tangent- and adjoint-mode AD. One reason for our choice is the direct access to the differentiated source, providing much flexibility and the opportunity to optimise the execution time of the adjoint by adjusting the differentiated program.

Collaboration between EDF and INRIA helped in applying TAPENADE to MASCARET and provided feedback and experience on the problems encountered, thus bringing improvements to both tools.

IV.4 Application of TAPENADE to MASCARET

IV.4.1 Context

As mentioned in section II.2, the search for an optimal friction coefficient during automatic calibration must be done by minimising the cost function (4). In our calibration experiments, we calibrate K defined as K_i for each friction zone i, whereas what the simulation uses is an expanded $expK$ defined at each node p. By definition, for all node p belonging to friction zone i:

$$expK_p = K_i \tag{8}$$

The cost function (4) becomes:

$$\hat{J}(K) = \sum_{c=1}^{N_{flood}} \left[\sum_{j=1}^{N_{meas}^c} \left(\alpha_j^c \left(Z_j^{meas} - Z_j^{calc}(expK(K)) \right)^2 \right) \right] \tag{9}$$

The gradient of $\hat{J}$ with respect to K is:

$$\frac{d\hat{J}}{dK} = \sum_{c=1}^{N_{flood}} \left[\sum_{j=1}^{N_{meas}^c} \left(-2\alpha_j^c \left(Z_j^{meas} - Z_j^{calc}(expK(K)) \right) \right) \frac{dZ_j^{calc}}{dexpK} \frac{dexpK}{dK} \right] \tag{10}$$

where Z_j^{calc} is the water surface profile calculated from $expK$ by the model. The matrix $\frac{dexpK}{dK}$ has a very simple structure, with one row per node and one column per friction zone. For each friction zone i, this column elements are 1 for nodes p belonging to i and 0 otherwise.

IV.4.2 Choice of the AD mode

The largest and most computationally-intensive part of equation (10) is the derivative of the water surface calculated at each point j of the grid as a function of the friction coefficients at each node:

$$\frac{dZ_j^{calc}}{dexpK} \tag{11}$$

As is usually the case, it is fortunately not needed to compute this large matrix explicitly. Equation (10) actually amounts to multiplying this matrix on the left and on the right as follows:

- on the left, it is multiplied with a single-row matrix, i.e. a transposed vector, whose element of rank j is:

$$\sum_{c=1}^{N_{flood}} -2\alpha_j^c (Z_j^{meas} - Z_j^{calc}(expK(K))) \tag{12}$$

  assuming there is a measurement available at each point. If not, set α_j^c to 0.

- on the right, it is multiplied with $\frac{dexpK}{dK}$ which has one column per friction zone. Our first applications have only a few friction zones.

As the left multiplier is a single-row matrix, we are aware that adjoint differentiation can compute the complete $\frac{d\hat{J}}{dK}$ in a single run of the adjoint code. On the other hand, since the right multiplier has one column per friction zone, the code produced by tangent differentiation must run once per friction zone to compute the full $\frac{d\hat{J}}{dK}$.

However, our first applications have less than 10 friction zones and experience shows that adjoint codes are almost always 2 to 5 times slower than tangent codes, due to their sophisticated architecture. Moreover, the present architecture of MASCARET doesn't lend itself easily to isolating the computation of the left row-vector multiplier. For these reasons, we feel it is wise to use the tangent mode of AD for our first calibration experiments, and in the future slightly refactor MASCARET and switch to the adjoint mode when it comes to larger cases.

IV.4.3 Actual differentiation and derivatives validation

Actual differentiation by TAPENADE of the steady subcritical kernel of MASCARET requires some technical adaptions, as is often the case for codes of this size (≈ 140000 lines of code). Differentiation itself produces a long message log, most of which can be discarded after a first careful look. The remaining messages are about limitations of the AD tool that must be worked around by modifying the source (e.g. array initializations), or limitations of AD that require post-processing of the differentiated code. In the latter category, differentiation of MASCARET introduces temporary arrays for intermediate variables and boolean masks, whose size could not be determined at differentiation time. The end-user is requested to provide these sizes in a special-purpose separate module.

As a research tool, TAPENADE also contains a number of errors. Collaboration between the AD tool developers and MASCARET developers is essential, resulting often in improvements to the AD tool and sometimes in clarifications to the MASCARET source.

When it finally compiles, the differentiated code must be incorporated in a calling context which is similar to that of the original code, except for additional variables that hold the input and output derivatives. This can be seen as the task of the final user but on this occasion we could test an experimental feature of the AD tool which generates a new calling context from the existing one by declaring, allocating, initializing and freeing the differentiated variables. This is an ongoing development, as adjoint differentiation of dynamic memory management is still an active research subject.

Once the differentiated code actually produces derivatives, it is necessary to check their correctness. Validation of the derivatives is usually performed in two steps:

- Validation of the tangent derivatives produced by AD, against derivatives approximated by Finite Differences.
- Validation of the adjoint derivatives against the (validated) tangent derivatives. Reusing the notation of section IV.1, given any two vectors δX and $\delta^* Y$, the scalar $\delta^* Y.\mathrm{A}.\delta X$ can be computed in two ways, either through tangent mode or through adjoint mode. The result must be the same up to machine precision.

We performed these two tests on MASCARET for the steady river test cases, with satisfying results. Even if for this work, we decided not to use the adjoint for the final calibration experiments, we did validate both tangent and adjoint codes:

- The maximum difference between the gradient calculated by FD and that obtained by AD is of the order of 10^{-4}.
- The maximum difference observed between the gradient calculated by the tangent and adjoint modes is of the order of machine precision ($\approx 10^{-14}$).

Finally, the runtime of the tangent code is roughly twice the run time or the original code, whereas the adjoint code is roughly seven times slower than the original code. This varies with the test case but conforms to what is commonly expected. This validates a posteriori our choice of using the tangent mode for the

following calibrations, and indicates to switch to the adjoint mode when calibrating a larger number of parameters.

V VALIDATION OF THE AUTOMATIC CALIBRATION METHOD

In order to validate the automatic calibration method developed during this work, two cases of real application were studied. Since the new algorithm allows bounding the search for optimal friction coefficients, in order to avoid all outliers and non-physical values, it is performed in 25 to 45 $m^{1/3}/s$ for the main channel and in 5 to 20 $m^{1/3}/s$ for the flood plain.

The results obtained with the method developed here were compared with those obtained using the old first order calibration method with a gradient calculated by FD. The comparison between the two approaches is performed by focusing on the speed of convergence of the different methods used, the final value of their cost function and, of course, the values of the friction coefficients obtained.

V.1 Calibration of the reach of the Rhone close to the Bugey nuclear power plant

V.1.1 Context and available data

This first case study corresponds to a 5 km section of the Rhone River located near the Bugey nuclear power plant. Upstream and downstream of the model, the boundary conditions used are the imposed flow rate and water level, respectively. In addition, for this case, 30 inflows distributed along the river were taken into account. The geometry of this calculation case is not described as it not the purpose of the present study.

Regarding the calibration, 8 friction zones of variable size were considered and 28 water level measures were available (cf. figure 1) for an upstream flow rate of 446 m^3/s and a downstream water level of 188.10 m. Since the upstream flow rate for which observations were available was relatively low, the flow regime studied was non-overtopping. Therefore, only the friction coefficients of the main channel could be calibrated.

Furthermore, since this case had already been the subject of a study, calibration data obtained manually were also available. The Strickler coefficient values per friction zone calibrated manually were (43 / 28.5 / 39 / 34 / 45 / 27 / 35 / 35) $m^{1/3}/s$.

V.1.2 Comparison of results obtained during automatic calibration

The purpose of this paragraph is to compare the results obtained with:

- The automatic calibration method using the BFGS optimisation method.
- The former calibration method using the gradient descent optimisation method.
- A reference manual calibration.

Figure 1 presents the water surface profiles obtained for these three different calibration methods. For the two automatic calibration methods, the Strickler coefficient was initialised at a value of 30 $m^{1/3}/s$ in each zone.

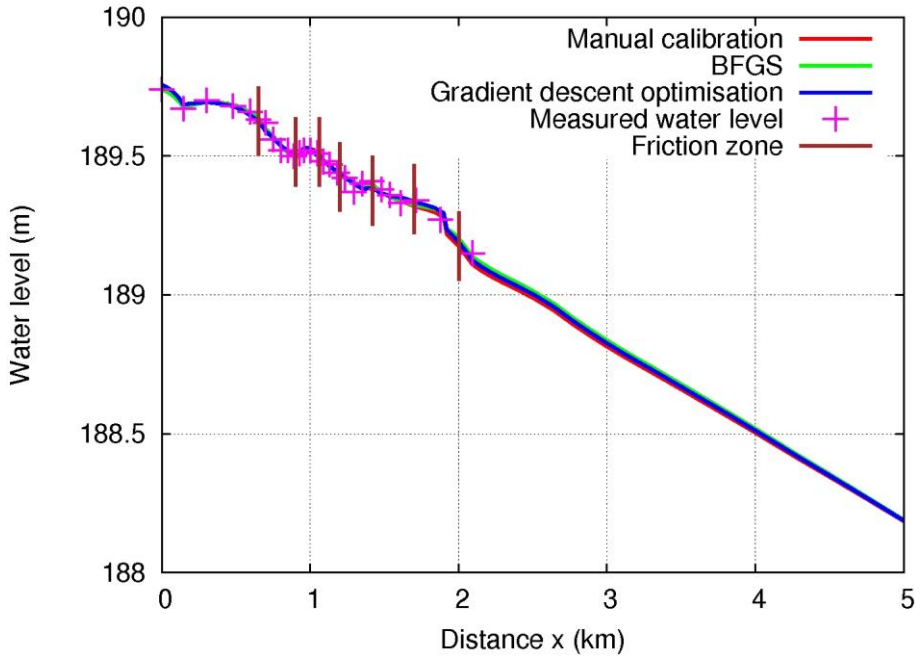

Figure 1: *Water height calculated using the friction coefficients of the main channel calibrated manually and automatically (BFGS, gradient descent optimisation).*

As shown in the figure above, the results obtained from the three calibration methods are very similar for the water surface profile calculated. This confirms the methodological choices that we present in this document.

Figure 2 shows the friction coefficient values obtained with the three methods.

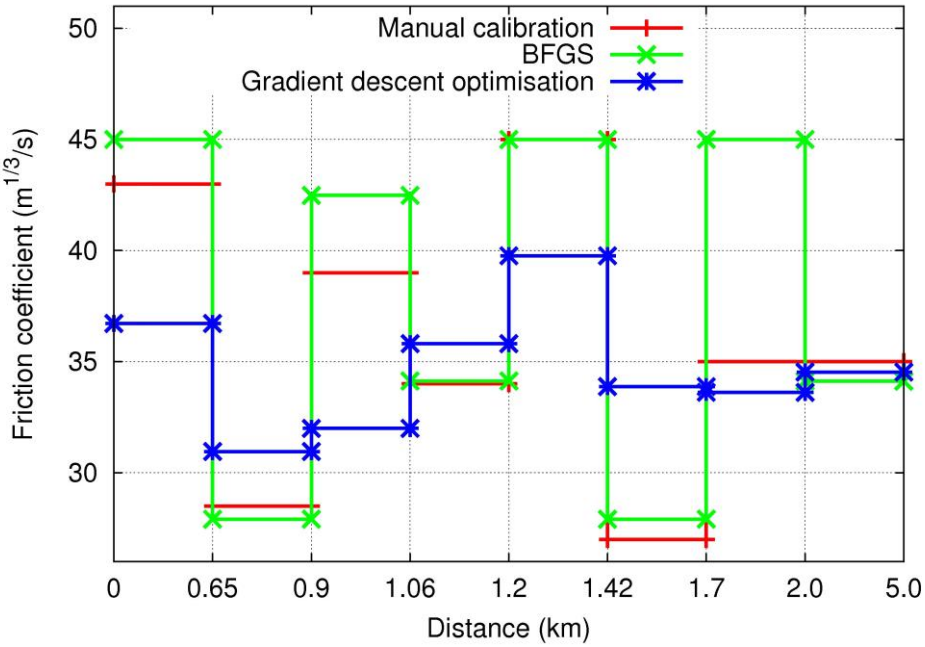

Figure 2: *Friction coefficient values by zone obtained with the three different calibration methods***.**

From Figure 2, we draw the following observations:

- The values resulting from the manual and automatic calibrations making use of the BFGS optimisation method are, generally speaking, quite close (about 10% in terms of relative error).
- The magnitude of variability of the Strickler coefficients between two consecutive friction zones is greater for the two previous methods than for automatic calibration method using the gradient descent optimisation method.

- The three calibration methods present an almost identical Strickler coefficient for the last friction zone.

The last comment permits understanding why, despite the different sets of friction coefficients, the water surface profiles calculated are quite close for the three calibration methods. It is very clear in the light of these results that the downstream friction zone is that which has the greatest impact on the calibration of this case study. This can be explained, on the one hand, by the size of this friction zone, which comprises about 3 km of the 5 km of the section studied; and on the other hand, by the resolution method used in the steady subcritical flow kernel of MASCARET which is performed from downstream to upstream for the calculation of the water surface profile along the length of the section studied. Furthermore, the results shown in Figure 2 emphasise the complexity of determining a set of optimal friction coefficients. Indeed, the uniqueness of a solution of this type of problem is not mathematically proven and different sets of parameters can give analogous results. Thus it is important to set bounds on the search for optimal friction coefficients to avoid all the outliers and non-physical values. This is why the automatic calibration method developed here uses a constrained optimisation approach.

Regarding the speed of convergence of the two automatic calibration methods (BFGS and gradient descent optimisation), Figure 3 highlights a faster convergence and accuracy for the automatic calibration method developed in this work in comparison to the old method. The BFGS method based on AD finds an optimal solution in less than 10 iterations whereas the old method reaches an equivalent result in 40 iterations. The manual calibration requires approximately 20 simulation runs to find a cost function between 10^{-1} and 10^{-2}.

For the total computation time, there is no noticeable difference to reach the same accuracy on the cost function for both methods.

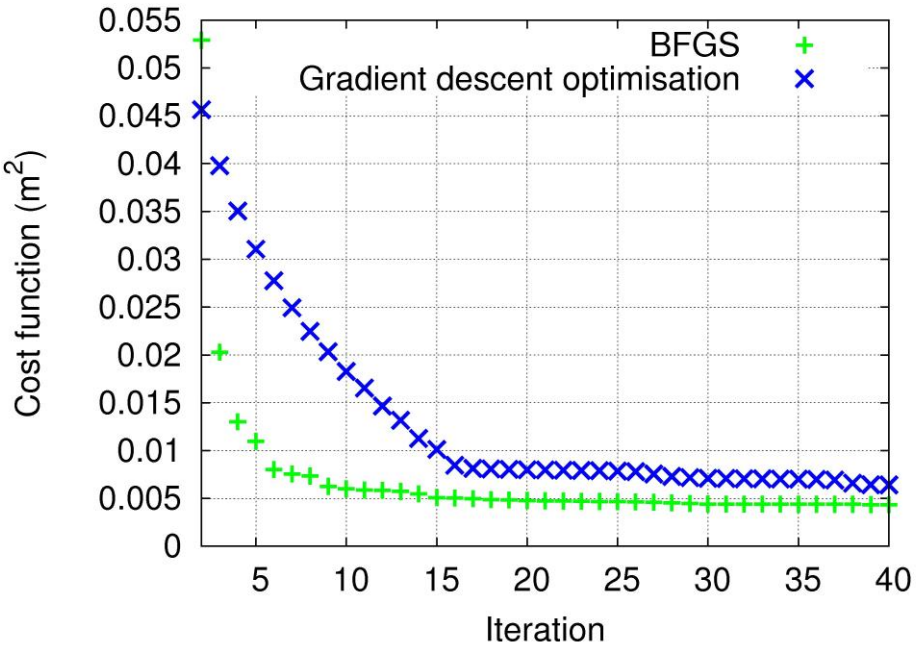

Figure 3: *Value of the cost function according to number of iterations for the Quasi-Newton BFGS method and the gradient descent optimisation method.*

To conclude this first case of implementation, the automatic calibration method based on the constrained optimisation approach of the Quasi-Newton method BFGS presented better accuracy (after 40 iterations, the cost functions are constant with a value 4.3×10^{-3} m^2 for the BFGS approach versus 6.4×10^{-3} m^2 for the gradient descent optimisation method) with faster convergence in comparison to the old method used.

To compare these initial findings, a second application case study was tested. It takes a 50 km section of the Garonne River and includes both the main channel and the flood plain.

V.2 Calibration of a reach of the Garonne

V.2.1 Context and available data

The zone selected to perform this study was a section of the Garonne River between Tonneins, downstream of the confluence with the Lot River, and La Réole (limit of the hydrodynamic influence of the tide), i.e. about 50 km of river [Besnard and Goutal, 2011].

Upstream and downstream of the model, the boundary conditions used were imposed flow rate and water surface profile, respectively. The section studied was divided into 3 friction zones of variable size (cf. Figure 4).

To calibrate this case, two sets of flood data each composed of 3 measures were available. The first set of data concerned a non-overtopping flood with a flow rate of 255 m^3/s and a downstream height of 4 m whereas the second set resulted from an overtopping flood with a flow rate of 2550 m^3/s and a downstream height of 11.73 m. The geometry of the model is not described in this paper.

Furthermore, since this case had already been the subject of a study, the calibration data obtained manually were also available. The Strickler coefficient values per friction zone calibrated manually were, for main channel and the flood plain, (40/32/33) $m^{1/3}/s$ and (10/12/12) $m^{1/3}/s$, respectively. The water surface profile linked to these friction coefficient values is shown in the figure below:

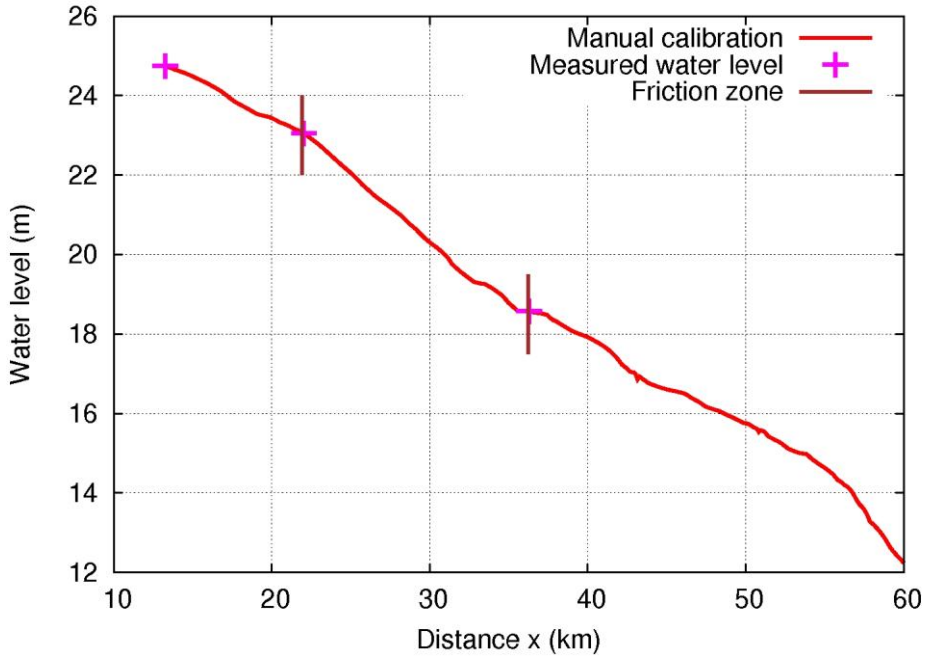

Figure 4: *Water surface profile calculated with friction coefficients calibrated manually.*

V.2.2 Comparison of results obtained during automatic calibration

For the two automatic calibration methods, the Strickler coefficients of the main channel and flood plain were initialised at the values of 30 $m^{1/3}/s$ and 15 $m^{1/3}/s$ respectively, in each zone. In this case, the automatic calibration determined the friction coefficients of the main channel and flood plain simultaneously with the two floods available.

The results obtained with the two automatic calibration methods are presented in the following figure:

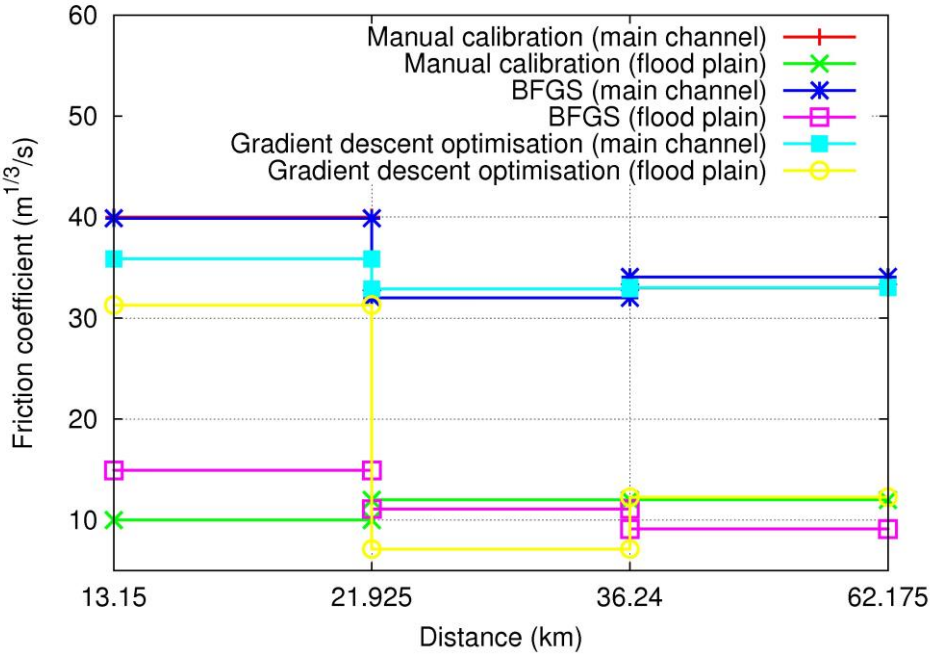

Figure 5: *Friction coefficient values of the main channel and flood plain obtained with different automatic calibration approaches***.**

The first two observations mentioned in the presentation of the results of the previous test case remain valid for this new application.

However, in comparison to the previous case, the results shown in Figure 5 highlight the need to set bounds when searching for the friction coefficient. Indeed, the friction coefficient value for a natural flood plain must be between 5 and 20 $m^{1/3}/s$. In this case of application, contrary to the automatic calibration method based on the constrained BFGS Quasi-Newton method, this criterion is not complied with in the calibration based on the gradient descent optimisation method, for which the friction coefficient of the flood plain which extends from 13150 m to 21925 m was found equal to 31 $m^{1/3}/s$.

Regarding the speed of convergence of the two automatic calibration methods (BFGS and gradient descent optimisation), Figure 6 also highlights higher accuracy and speed of convergence for the number of iterations, for the automatic calibration based on the Quasi-Newton BFGS optimisation method than for the old method. After only 9 iterations, the cost function was 7×10^{-6} m^2 for the BFGS method versus 0.14 m^2 for the gradient descent optimisation method. This last value is too high to consider the found Strickler coefficients as acceptable.

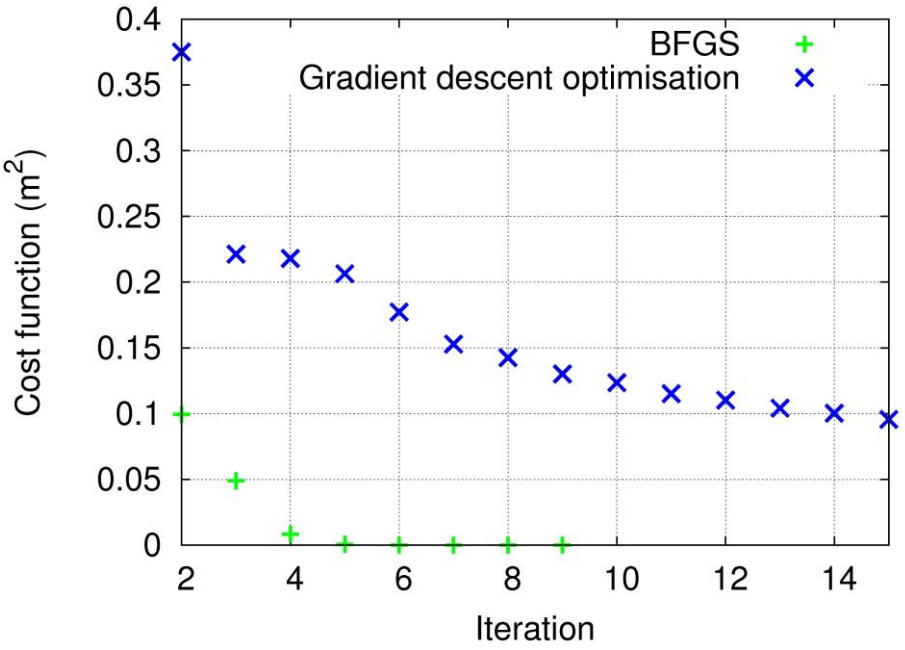

Figure 6: *Cost function values according to the number of iterations obtained with the different automatic calibration approaches.*

The case of the Garonne emphasised the efficiency of the constrained BFGS Quasi-Newton method when calibrating friction coefficients in the framework of a river, on the one hand by bounding the minimisation and on the other hand providing better accuracy and convergence speed.

V.3 Synthesis: contribution of AD to the automatic calibration

The real test cases of Bugey and Garonne show the superiority of the second order BFGS method over the first order gradient method as expected. Even if the cost function of the BFGS method is always smaller than for the gradient method, it is not easy to set a value of convergence criteria for the cost function from a practical point of view. This mainly depends on the number of floods/measurements and on the required accuracy for the computed water levels. Consequently, practical applications can have a wide range of values for the convergence criteria.

Finally, in order to see more clearly the gain of using AD, the BFGS method is tested on a new set of data with FD. The following table compares the cost function values for 15 iterations of the BFGS algorithm.

Table 2: Influence of the derivative method on the BFGS algorithm

BUGEY		**GARONNE**	
FD	**AD**	**FD**	**AD**
6.09×10^{-3}	4.21×10^{-4}	6.39×10^{-1}	3.98×10^{-10}

VI CONCLUSIONS AND OUTLOOK

Evaluating derivatives for a given function (mathematical function and calculation code) can be a challenge. Algorithmic Differentiation proves a powerful technique for evaluating the derivatives of functions described by computer programs [Griewank and Walther, 2008].

This article presented the application of the AD tool TAPENADE [Hascoët and Pascual, 2013] to one of the hydraulic codes of MASCARET [Goutal et al., 2012], and validated its use for an inverse parametric optimisation problem. For 1D hydraulic programs like MASCARET, the nature of a river bed is modelled by a friction coefficient. This coefficient accounts for the friction of walls on fluid as well as other phenomena not modelled elsewhere such as turbulence and channel bends. The objective of this work was to apply AD to the inverse problem of optimal friction coefficient calibration.

Automatic calibration is an inverse method used to obtain a constant “admissible” friction coefficient per zone, resulting in the calculation of a water surface profile close to that measured for a steady flow. The

optimal search for this coefficient takes the form of minimising a cost function calculating the difference between the height calculated by the numerical model and the measured height. This led to choosing the constrained BFGS Quasi-Newton method to minimise the cost function. Using a constrained optimisation method allowed setting bounds when searching the parameter to be calibrated. The optimisation method employed involved calculating the gradient of the cost function, which was obtained through AD of the calculation code.

The results obtained with the method developed were compared with those obtained with an old calibration method based on the gradient descent optimisation method with a gradient approximated by FD. The comparison was performed on two real case studies.

These two cases of application highlighted the efficiency of the constrained BFGS Quasi-Newton method during the calibration of friction coefficients in comparison to gradient descent optimisation due to the setting of bounds for minimisation. Indeed, searching an optimal friction coefficient calibration is a complex, sometimes ill-posed problem for which different sets of parameters can provide analogous results. This question has not been investigated here, but to avoid any outliers or nonphysical values, it is important to set bounds on the search for optimal friction coefficients. Furthermore, in the light of the different results presented, the speed of convergence of the new calibration method is faster in terms of iterations and better accuracy is obtained. The automatic calibration method developed during this work will therefore be deployed in the upcoming version of the MASCARET software (version 8.1).

For several decades, EDF R&D has developed numerical codes to respond to the problems encountered by the company. Innovative techniques such as AD may fit into the constant improvement of these tools.

AD covers a wide array of applications. This work was conducted in the framework of inverse problems. However, other applications will also be explored:

- Quantification of uncertainty. Certain methods allow exploring the range of variation of calculation code inputs using partial derivatives to deduce global sensitivity indices.
- Variational data assimilation methods ("4DVAR" algorithm). As with automatic calibration, this type of algorithm requires the use of an optimisation method which in turn requires calculating the gradient of a cost function.

VII REFERENCES

Besnard A., Goutal N. (2011) – Comparison between 1D and 2D models for hydraulic modeling of a floodplain: case of Garonne river. *La Houille Blanche*, **3**: 42-47

Bischof C. H., Roh L., Mauer-Oats A. J. (1997) – ADIC: an extensible automatic differentiation tool for ANSI-C. *Software: Practice and Experience*, **27(12)**: 1427-1456.

Bischof C. H., Carle A., Corliss G. F., Griewank A., Hovland P. D. (1991) – ADIFOR – Generating Derivative Codes from Fortran Programs. *Scientific Programming*, **1**: 11-29.

Chanson H. (2004) – *Environmental hydraulics of open channel flows.* Elsevier Butterworth-Heinemann.

Cunge J. A., Holly F. M., Verwey A. (1980) – *Practical aspects of computational river hydraulics*. Pitman Advanced Publishing Program.

Fread D. L., Smith G. F. (1978) – Calibration technique for 1-D unsteady flow models. Journal of Hydraulics Division. *Proceedings of the ASCE*, **104(HY7)**: 1027-1044

Giering R., Kaminski T., Slawig T. (2005) – Generating efficient derivative code with TAF: Adjoint and Tangent linear Euler flow around an airfoil. *Future Generation Computer Systems* **21(8)**: 1345-1355

Gilbert J.-C., Lemaréchal C. (1989) – Some numerical experiments with variable-storage quasi-Newton algorithms. *Mathematical Programming*, **45(1-3)**: 407-435

Goutal N., Lacombe J.-M., Zaoui F., El Kadi Abderrezzak K. (2012) – MASCARET : a 1-D open-source software for flow hydrodynamic and water quality in open channel networks. *River Flow, Murillo (Ed.), Taylor & Francis Group, London*, 1169-1174.

Griewank A., Walther A. (2008) – *Evaluating derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics.

Hascoët L., Pascual V. (2013) – The Tapenade Automatic Differentiation tool: Principles, Model and Specification. *ACM Transactions on Mathematical Software*, **39(3)**: 20:1-20:43.

Lotz J., Leppkes K., Naumann U. (2011) – *dco/c++ - Derivative Code by Overloading in C++*. Aachener Informatik-Berichte (AIB)

Naumann U. (2012) – *The Art of differentiating Computer Programs. An Introduction to Algorithmic Differentiation.* SIAM-Society for Industrial and Applied Mathematics.

Nocedal J., Wright S. J. (2006) – *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer.

Utke J., Naumann U., Fagan M., Tallent N., Strout M., Heimbach P., Hill C., Wunsch C. (2008) – OpenAD/F: A modular, open-source tool for Automatic Differentiation of Fortran codes. *ACM Transactions on Mathematical Software*, **34(4)**: 18:1-18:36.

Walther A., Griewank A. (2012) – Getting started with ADOL-C. *Combinatorial Scientific Computing. U. Naumann and O. Schenk (Ed.), 181-202*